Brad Abrams

Designing Microsoft® .NET Class Libraries

June 2004

Lesson 1: Naming Conventions

- After successfully completing this lesson, you will be able to:
  - Use correct naming for all publicly exposed identifiers

Naming Patterns

- Only for publicly exposed identifiers
  - Just what we document
  - NOT a source coding convention
- PascalCasing: Each word starts with an uppercase letter
- camelCasing: First word is lowercase, others uppercase

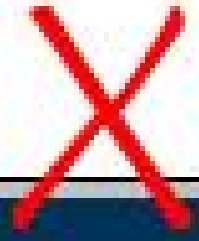
Naming Patterns

- All types and publicly exposed members are PascalCased
- Parameters are camelCased
  - These show up in Microsoft® IntelliSense® and docs
- Abbreviations that are longer than two letters are cased as words, otherwise ALLUPPER
  - IO vs. Html

Hungarian Notation

- Do not use Hungarian notation in publicly exposed APIs and parameter names

```
public class CMyClass {
    int CompareTo (object objValue) {..}
    string lpstrName {get;}
    int iValue {get;}
}
```



Hungarian Notation (continued)

- The prefix codes are arbitrary and difficult to learn
  - They make it harder to read an identifier name
- Hungarian was developed for a time when languages were loosely typed and IDEs were not as powerful
  - Example: "out" and "ref" parameters information now conveyed at the call site

```
Interlocked.Increment(ref i)
```

Naming Patterns

- Identifiers should be readable
  - Do not use underscores
  - Do not use all caps
  - Do not use Hungarian
  - Properties that return arrays\collections are plural
    - Example: Type.GetMembers()
- Instead of lpcUserName use userName
- Instead of MAX_VALUE use MaxValue

Naming Patterns (continued)

- Familiar to many users
- Used consistently in the Framework

```
public class MemberDoc {
    int CompareTo (object value) {..}
    string Name {get;}
}
```

Class and Interface Naming

- Do not prefix class names with "C"
  - Class is the default
  - Noise
- Do prefix Interface names with "I"
  - Makes them stand out—different from classes

```
public class Component {}
public interface IFormattable {}
```

Naming Issues

- Good naming is extremely hard
  - Be meaningful but brief
- Use U.S. English
  - Example: Colour vs. Color
- Avoid abbreviations
  - Use the "Google Test"
- Consider the principle of least surprise
- Look for prior-art
  - Ex: NumberOfElements vs. Count

Type Naming

- Choose type names that consist of nouns or noun phrases—examples: LegalDocument, FileSystem etc.
- If you're deriving from core types, add a suffix to identify the type:
  - Argument*Exception*
  - File*Stream*
  - Click*EventArgs*

Method Naming

- Methods are means of taking action
- Use verbs or verb phrases
  - Also helps distinguish methods from type names

```
public void Remove (object item)

protected void GetObjectData(…)
```

Parameter Naming

- Parameter names are displayed in visual design tools, IntelliSense, and class browsers
- Use descriptive names based on the parameter's *meaning* rather than the parameter's *type*
  - Example: "maxValue" rather than "i"
- Use camelCase

Exercise: Fixing a Broken Class Definition

- Fix this class definition to conform to naming conventions:

```
public class HtmlEncoding {
  public const string DEFAULT_NAME = "Html3.2";
  public HtmlEncoding (int iCount) {..}
  public string TagName {get {..}}
  public bool UseIoCompletionPort {get {..}}
  protected void _coreSetValue (double value)
  {..}
  private IntPtr ptrValue;
}
```

Lesson 1 Summary

- A consistent naming pattern is important
- Identifier readability is key
- Use PascalCasing for types and publicly exposed members
- Use camelCasing for parameter names
- Interfaces are prefixed with "I"
- Class are NOT prefixed with "C"